CEYX-Version 15.III : VPRINT le composeur CEYX

Gérard Berry, Jean-Marie Hullot

▶ **To cite this version:**

Gérard Berry, Jean-Marie Hullot. CEYX-Version 15.III : VPRINT le composeur CEYX. [Rapport Technique] RT-0046, INRIA. 1985, pp.13. inria-00070112

HAL Id: inria-00070112
https://inria.hal.science/inria-00070112

Submitted on 19 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tel (3) 954 90 20

Rapports Techniques

N° 46

CEYX - Version 15

III : VPRINT, LE COMPOSEUR CEYX

Gérard BERRY
Jean-Marie HULLOT

Février 1985

INRIA
Domaine de Voluceau
Rocquencourt
78153 Le Chesnay Cedex

CEYX - Version 15

III: VPRINT, Le Composeur CEYX

Gérard Berry, Jean-Marie Hullot

Eté 1984

Résumé: *Nous définissons une machine virtuelle permettant de produire des représentations textuelles de structures LISP ou CEYX arbitraires.*

Abstract: *We define a virtual machine intended to produce textual representations for any LISP or CEYX structure.*

PAPIER RECUPERE ET RECYCLE

VPRINT

Le Composeur CEYX

Gérard Berry, Jean-Marie Hullot

Nous définissons une machine pour composer CEYX. Cette documentation donne les fonctions de base de la machine et un programme standard de paragraphage (pretty-print) utilisant cette machine.

La lecture de ce document suppose une connaissance préalable de CEYX et LE_LISP. Nous renvoyons pour cela aux rapports INRIA suivants:

- LE_LISP de l'INRIA, Le Manuel de Référence (J. Chailloux),
- CEYX: Une Initiation (J.-M. Hullot),
- Programmer en CEYX (J.-M. Hullot).

1 La Machine de Composition

1.1 Impression des Atomes

(with-vprint-output <expr>1 ... <expr>n)

Les <expr>i sont des expressions LISP quelconques qui sont évaluées en séquence. Toutes les fonctions d'impression et de composition décrites ci-dessous ne fonctionnent que si elles sont utilisées dans le corps d'un with-vprint-output.

(vpatom <atom>)

Permet d'imprimer un atome LISP quelconque.

(vprincn <charn>)

<charn> est un code ascii, cette fonction imprime le caractère correspondant.

(vprinch <char>)

<char> est un symbole monocaractère qui est imprimé.

Exemples:

```
? (with-vprint-output (vprinch 'a))
a
= t
? (with-vprint-output (vprincn #/a))
a
= t
? (with-vprint-output (vpatom 'Celui) (vprinch '| |)
?                     (vpatom "qui") (vprinch '| |)
?                     (vpatom "sait") (vprinch '| |)
                      (vpatom "ne") (vprinch '| |)
                      (vpatom "parle") (vprinch '| |)
                      (vpatom "pas") (vprincn #/.) (vprinch '| |))
Celui qui sait ne parle pas.
= t
```

1.2 Les Blocs de Composition

Les fonctions de la section précédente permettent de construire des ***blocs de texte*** monolignes. Pour l'exemple précédent, nous avons le bloc de texte:

```
•---------------------------•
|Celui qui sait ne parle pas.|
•---------------------------•
```

Dans le cas où ce bloc tient dans une ligne physique (i.e. sa longueur est inférieure à (rmargin)), il est imprimé tel quel. S'il ne tient pas dans une ligne, aucune indication n'est donnée sur la façon dont on doit le couper en plusieurs lignes lors de l'impression. Nous dirons que les blocs de texte sont ***insécables***.

Nous donnons maintenant des constructions permettant de structurer des blocs de manière à autoriser l'impression sur plusieurs lignes. Pour cela, nous introduisons les notions de ***point de coupure*** et de ***bloc de composition***.

(vcutpoint <atom>)
(vcutpoint)

Cette fonction introduit un point de coupure optionnel qui est interprété de manière différente dans le contexte d'un bloc horizontal, vertical ou mixte (voir ci-dessous).

1.2.1 Les Blocs Horizontaux

(hblock <indent> <expr>1 ... <expr>n)

<indent> doit avoir pour valeur un entier. Les <expr>i sont des expressions LISP quelconques qui sont évaluées en séquence. Cette fonction place la machine de composition en mode horizontal.

```
(hblock 3
    (vpatom "Celui")
    (vcutpoint)
    (vpatom "qui")
    (vcutpoint)
    (vpatom "sait")
    (vcutpoint)
    (vpatom "ne")
    (vcutpoint)
    (vpatom "parle")
    (vcutpoint)
    (vpatom "pas")
    (vprincn #/.))
```

```
•-----------------------------------------•
|•-----• •---• •----• •--• •-----• •----•|
||Celui|^|qui|^|sait|^|ne|^|parle|^|pas.||
|•-----• •---• •----• •--• •-----• •----•|
•-----------------------------------------•
```

En mode horizontal, les points de coupure sont interprétés comme des espaces tant que la marge droite n'est pas atteinte, si aucun argument n'est donné à vcutpoint. Si un argument <atom> est donné, on imprime cet atome à la place de l'espace. Ainsi si toute l'expression tient sur la ligne, le résultat de l'impression est:

```
Celui qui sait ne parle pas.      |<-- rmargin
```

Dans le cas contraire, un saut de ligne est produit à la place du dernier point de coupure sur la ligne, et l'impression reprend à la ligne suivante avec le renfoncement <indent>:

```
Celui qui sait  |<-- rmargin
   ne parle pas.|
```

ou, avec une marge droite plus petite,

```
Celui qui     |<-- rmargin
```

```
sait ne   |
parle pas.|
```

Noter que dans ce cas le blanc ou la chaîne argument de vcutpoint n'est pas imprimé.

{VPRINT}:indent [variable]

La variable {VPRINT}:indent contient la valeur par défaut du renfoncement. A l'initialisation du système, elle vaut 3 et elle peut bien sur être modifiée par l'utilisateur.

1.2.2 Les Blocs Verticaux

(vblock <indent> <expr>1 ... <expr>n)

<indent> doit avoir pour valeur un entier. Les <expr>i sont des expressions LISP quelconques qui sont évaluées en séquence. Cette fonction place la machine de composition en mode vertical.

Avec l'exemple précédent, dans lequel hblock est changé en vblock,

```
(vblock 3 (vpatom "Celui") (vcutpoint) ...)
```

nous avons systématiquement la composition verticale suivante:

```
Celui
   qui
   sait
   ne
   parle
   pas.
```

c'est à dire que tous les points de coupure sont interprétés comme des sauts de ligne.

1.2.3 Les Blocs Mixtes

(xblock <indent> <expr>1 ... <expr>n)

<indent> doit avoir pour valeur un entier. Les <expr>i sont des expressions LISP quelconques qui sont évaluées en séquence. Cette fonction place la machine de composition en mode mixte.

Reprenons l'exemple précédent dans le cas d'un bloc mixte:

```
(xblock 3 (vpatom "Celui") (vcutpoint) ...)
```

Ici la composition se fait en mode horizontal si tout tient sur une seule ligne et en mode vertical dans le cas contraire:

```
Celui qui sait ne parle pas      |<-- rmargin
```

ou:

```
Celui             |<-- rmargin
   qui            |
   sait           |
   ne             |
   parle          |
   pas.           |
```

Comme précédemment, (vcutpoint <atom>) imprime l'atome à la place du blanc en mode horizontal, et provoque un retour chariot sans impression de l'atome en mode vertical. De plus un appel de vcutpoint ne provoque pas de retour chariot si rien n'a été écrit dans la ligne courante, afin d'éviter la ligne blanche qui en résulterait.

1.2.4 Exemples

Etant donnée une liste d'atomes L, nous définissons la fonction qui permet d'imprimer cette liste en insérant un point de coupure entre chacun des éléments de la liste:

Nous construisons deux listes que nous allons utiliser pour visualiser le résultat de la composition. La première liste L1 tient toujours sur une seule ligne, la liste L2 demande plusieurs lignes pour s'imprimer:

```
? (setq L1 '(<item>1 <item>2 <item>3))
= (<item>1 <item>2 <item>3)
? (setq L2 '(<item>1 <item>2 <item>3 <item>4 <item>5 <item>6 <item>7 <item>8
?           <item>9 <item>10 <item>11 <item>12 <item>13 <item>14 <item>15))
= (<item>1 <item>2 <item>3 <item>4 <item>5 <item>6 <item>7 <item>8 <item>9
<item>10 <item>11 <item>12 <item>13 <item>14 <item>15)
```

Nous nous plaçons d'abord dans un mode de composition horizontale:

```
? (with-vprint-output (hblock 0 (print-list-atoms L1)))
<item>1 <item>2 <item>3
= t
```

Les différents atomes apparaissent séparés par des espaces qui sont l'interprétation des points de coupure dans ce cas.

Toujours en mode horizontal, mais dans le cas où une ligne ne suffit pas pour l'impression, nous avons:

```
? (with-vprint-output (hblock 0 (print-list-atoms L2)))
<item>1 <item>2 <item>3 <item>4 <item>5 <item>6 <item>7 <item>8 <item>9
<item>10 <item>11 <item>12 <item>13 <item>14 <item>15
= t
```

L'atome <item>10 ne peut dans ce cas tenir sur la ligne; le point de coupure entre <item>9 et <item>10 est alors interprété comme un saut de ligne, les autres continuant à être interprété comme des espaces. Le renfoncement associé au bloc horizontal vaut 0, <item>10 commence en début de ligne. Nous donnons ci-dessous un exemple avec un renfoncement non nul:

```
? (with-vprint-output (hblock 8 (print-list-atoms L2)))
<item>1 <item>2 <item>3 <item>4 <item>5 <item>6 <item>7 <item>8 <item>9
        <item>10 <item>11 <item>12 <item>13 <item>14 <item>15
= t
```

Nous reprenons maintenant les mêmes exemples dans un mode de composition verticale:

```
? (with-vprint-output (vblock 0 (print-list-atoms L1)))
<item>1
<item>2
<item>3
= t
```

Dans ce cas, tous les points de coupure sont interprétés comme des sauts de lignes. Avec un renfoncement non nul, nous avons:

```
? (with-vprint-output (vblock 3 (print-list-atoms L1)))
<item>1
   <item>2
   <item>3
= t
```

Enfin, dans le cas d'un mode de composition mixte, le mode de composition horizontale est choisi si l'impression peut se faire sur une seule ligne et le mode de composition verticale si une ligne n'est pas suffisante:

```
? (with-vprint-output (xblock 0 (print-list-atoms L1)))
<item>1 <item>2 <item>3
= t
? (with-vprint-output (xblock 0 (print-list-atoms L2)))
<item>1
<item>2
<item>3
<item>4
<item>5
<item>6
<item>7
<item>8
<item>9
<item>10
<item>11
<item>12
<item>13
<item>14
<item>15
= t
```

Pour les initiés, remarquons que dans ce dernier mode il est préférable d'introduire un point de coupure dès le début pour éviter les interférences avec les blocs précédents, La bonne fac,on d'imprimer une liste en xblock est ainsi :

```
? (de print-list-atoms-xblock (l)
?     (xblock 0
?        (vcutpoint "")
?        (vpatom (nextl l))
?        (while l
?            (vcutpoint)
?            (vpatom (nextl l)))))
= print-list-atoms-xblock
```

1.3 Autres Commandes

(vterpri)

Cette fonction déclenche un saut de ligne inconditionnel.

(vindent <indent>)

<indent> est un entier, cette fonction augmente le renfoncement du bloc de composition courant de la valeur <indent>.

(begin-hblock)

Cette fonction déclenche l'ouverture d'un bloc horizontal qui devra être fermé ultérieurement par (end-block). La construction hblock est définie à l'aide de cette fonction par:

```
(defmacro hblock (indent . body)
    `(progn
        (begin-hblock)
        (vindent ,indent)
        ,@body
        (end-block)))
```

(begin-vblock)

Identique à la précédente pour les blocs verticaux.

(begin-xblock)

Identique à la précédente pour les blocs mixtes.

(end-block)

Ferme un bloc de quelque nature qu'il soit.

2 Le Programme Standard d'Impression

En plus de la machine de composition présentée dans la section précédente, nous donnons un programme standard d'impression pour les expressions LISP et CEYX. Au moment de l'écriture de cette documentation (Septembre 1984), le nombre de formats de composition définis en standard n'est pas suffisant pour obtenir un paragraphage idéal de toutes les expressions LISP. Ceci sera complété avec le temps.

2.1 Fonctions Générales

(vprint <expr> [profondeur])

Cette fonction réalise un paragraphage (pretty-print) de l'expression <expr>, jusqu'à la profondeur donnée en second argument ou jusqu'à la profondeur standard contenue dans la variable {VPRINT}:level (valant par défaut (printlevel)). Ceci signifie que l'impression s'arrête à la profondeur dite, le symbole # étant alors imprimé à la place de la sous-expression correspondante.

```
? (vprint '(a b c))
(a b c)
= t
? (vprint '((a b)(c d)))
((a b) (c d))
= t
? (vprint '((a b)(c d)) 1)
(# #)
= t
```

(vpretty <function-name> [profondeur])

Cette fonction réalise le paragraphage (pretty-print) de la fonction de nom <function-name>.

```
? (vpretty print-list-atoms)
(de print-list-atoms (l)
    (vpatom (nextl l)) (while l (vcutpoint) (vpatom (nextl l))))
= t
```

2.2 Contrôle du format d'impression par l'utilisateur

La fonction vprint peut être définie en LISP par:

```
(de vprint (x)
    (let ((x (car args))
          ({VPRINT}:level (if (cadr args) (cadr args) {VPRINT}:level)))
         (with-vprint-output (vprin x))))
```

et c'est la fonction vprin qui se charge de la composition suivant les cas.

(vprin <expr>)

Cette fonction se charge d'aiguiller la composition de l'expression <expr>. Elle peut être définie en LISP par:

```
(de vprin (x)
   (let (({VPRINT}:curlevel (1+ {VPRINT}:curlevel)))
    (if (> {VPRINT}:curlevel {VPRINT}:level)
        (vprincn #/#)
        (cond
          ((tconsp x) (sendq vprin x))
          ((null x) (vpatom "()"))
          ((stringp x) (vprincn #/") (vpatom x) (vprincn #/"))
          ((atom x) (vpatom x))
          ((consp x)
           (if (symbolp (car x))
               (selectq (ptype (car x))
                    (1 ({Vformat}:progn x))
                    (2 ({Vformat}:if x))
                    (3 ({Vformat}:defun x))
                    (4 ({Vformat}:cond x))
                    (5 ({Vformat}:selectq x))
                    (6 ({Vformat}:setq x))
                    (t (let ((vformat (getprop (car x) 'vformat)))
                            (ifn vformat
                                 ({Vformat}:data x)
                                 (funcall vformat x)))))
               ({Vformat}:data x)))
          (t (syserror 'vprin "Type LISP Inconnu"))))))
```

Pour les objets atomiques, le comportement de cette fonction est clair, nous nous attachons dans la suite au cas des objets qui sont des tcons ou des cons.

2.2.1 Composition des Objets Taggés (tcons)

Dans le cas où elle rencontre un objet taggé, la fonction vprin envoie le message vprin à l'objet en question.

Définissons par exemple le Trecord toto par:

```
? (deftrecord toto a b c)
= toto
? (setq x (omakeq toto a 1 b 2 c 3))))
= #(toto 1 2 . 3)
```

Si nous ne définissons pas de sémantique vprin pour toto, la sémantique vprin du modèle * est récupérée, on obtient ainsi une composition identique à celle fournie par LE_LISP:

```
? (vprint x)
#(toto 1 2 . 3)
= t
```

puisque {*}:vprin est défini par:

```
? (de {*}:vprin (x) (prin x))
= #:*:vprin
```

L'utilisateur peut changer ce format en redéfinissant la sémantique vprin pour le modèle toto:

```
? (de {toto}:vprin (x)
?     (vblock 3
?        (vpatom 'toto)
```

```
?       (vcutpoint)
?       (vpatom "a: ")
?       (vprin ({toto}:a x))
?       (vcutpoint)
?       (vpatom "b: ")
?       (vprin ({toto}:b x))
?       (vcutpoint)
?       (vpatom "c: ")
?       (vprin ({toto}:c x))))
= #:Model:Tmodel:toto:vprin
? (vprint x)
toto
   a: 1
   b: 2
   c: 3
= t
```

2.2.2 Composition des Objets de Type Cons

Conformément à l'usage dans les pretty-printer LISP, les objets de type cons sont composés suivant la valuer de leur car. Dans le cas où ce car n'est pas un atome on appliquera le format par défaut (i.e. {Vformat}:data), sinon on regarde si un format n'a pas été attaché à cet atome. Pour ce faire nous regardons tout d'abord leur ptype à la LE_LISP et sinon nous regardons sur la pliste de cet atome si une fonction de composition leur a été attaché sous la propriété vformat.

Pour associer une fonction de composition à un atome, nous définissons la fonction deformat:

(deformat <symbol> (<arg>) . <body>))

Cette construction définit la fonction {Vformat}:<symbol>, et attache le symbole {Vformat}:symbol sur la pliste de <symbol> sous l'indicateur vformat de manière à ce qu'il puisse être récupéré par vprin:

```
(de {Vformat}:symbol (<arg>) .@<body>)
```

Dans le cas où l'appel à la forme (deformat <symbol>1 <symbol>2) {Vformat}:<symbol>2 est attaché à <symbol>1 sous l'indicateur vformat.

Exemples:

Le format standard est défini par:

```
(deformat data (l)
    (hblock 1 (vprincn #/() (vplist l) (vprincn #/))))
```

Le format 'progn' est défini par (nous rappelons que la variable {VPRINT}:indent contient un entier qui est le renfoncement par défaut):

```
(deformat progn (l)
    (xblock {VPRINT}:indent
            (vprincn #/()
            (vplist l)
            (vprincn #/))))
```

où vplist est définie par:

(vplist <lexpr>)

```
(de vplist (l)
    (vprin (nextl l))
    (while (consp l) (vcutpoint) (vprin (nextl l)))
    (when l
          (vcutpoint)
```

```
            (vprincn #/.)
            (vcutpoint)
            (vprin 1)))
```

Le format 'setq' est défini par:

```
(deformat setq (l)
    (xblock {VPRINT}:indent
            (vprincn #/()
            (vprin (nextl l))
            (while l
                   (vcutpoint)
                   (xblock 2
                           (vprin (nextl l))
                           (vcutpoint)
                           (vprin (nextl l))))
            (vprincn #/))))
```

Nous aurons ainsi:

```
? (rmargin 79)
= 79
? (vprint '(a b c d e f g h i j))
(a b c d e f g h i j)
= t
? (vprint '(progn a b c d e f g h i j))
(progn a b c d e f g h i j)
= t
? (vprint '(setq a 1 b 2 c 3 d 4 e 5))
(setq a 1 b 2 c 3 d 4 e 5)
= t
? (rmargin 10)
= 10
? (vprint '(a b c d e f g h i j))
(a b c d e
 f g h i j)
= t
? (vprint '(progn a b c d e f g h i j))
(progn
   a
   b
   c
   d
   e
   f
   g
   h
   i
   j)
= t
? (vprint '(setq a 1 b 2 c 3 d 4 e 5))
(setq
   a 1
   b 2
   c 3
   d 4
   e 5)
= t
```

Index des fonctions LISP

Table des matières

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique